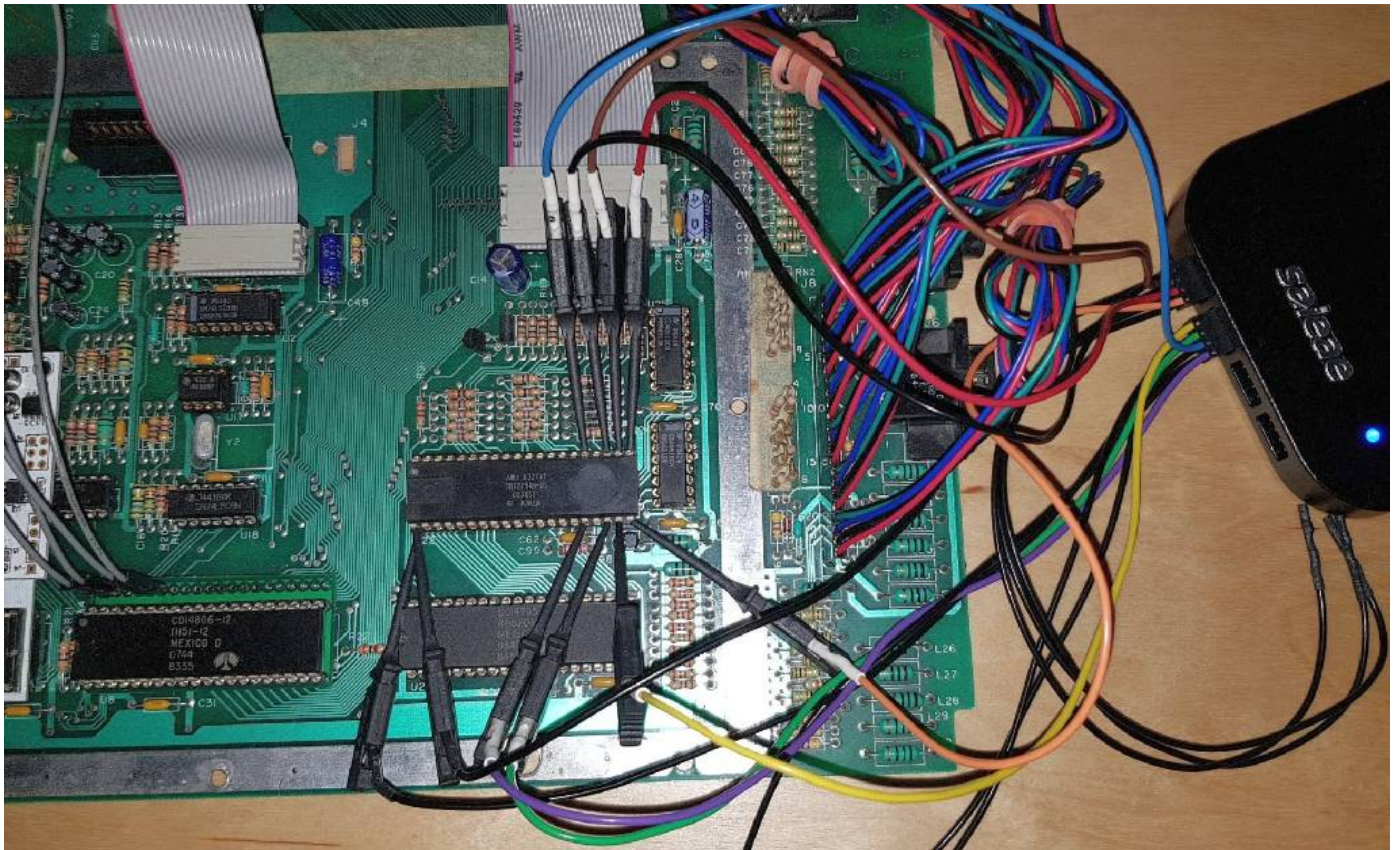


# Google CTF 2018 Quals - Wired CSV [Misc, 220p, 42 solver]

by *!SpamAndHex*


'Wired CSV' is clearly a hardware challenge: we were given a photo of some electronic device wired to a logic analyzer and the measurement data from the logic analyzer in CSV format.



First of all we have to identify the device under measurement. The top of the chip reads 'CO12294B' and after a little searching on the internet it is sure that this board is a part of an Atari computer. According to [ <https://www.atarimax.com/jindroush.atari.org/achip.html> ] the chip belong to a 1200xl, 800xl or 600xl version Atari. Further searching for each of them, on a forum [ <http://atariage.com/forums/topic/170558-help-atari-800xl/> ] a very similar board can be seen (slight difference is the main CPU, which is seems to have been prepared at the given one).

Fortunately for those, old devices it is very easy to find a huge range of scanned documents. The most useful among the manuals is the "Service Manual" [ [http://www.atarimania.com/documents/Atari\\_800XL\\_Sams\\_Computerfacts\\_Technical\\_Service.pdf](http://www.atarimania.com/documents/Atari_800XL_Sams_Computerfacts_Technical_Service.pdf) ] ( [http://www.atarimania.com/documents/Atari\\_800XL\\_Sams\\_Computerfacts\\_Technical\\_Service.pdf](http://www.atarimania.com/documents/Atari_800XL_Sams_Computerfacts_Technical_Service.pdf) ). We can witness many form of this manual, but those all contained a schematic of the PCB! (Unfortunately providing the schematic with the device is unimaginable for today device :( ) The service manual describes the chip we are interested in: it is a POKEY chip, which is responsible for reading the keyboard

input and outputting audio. Here is the pinout for the chip (taken from here: [[http://www.atarimania.com/documents/atari\\_800XL\\_Field\\_Service\\_manual.pdf](http://www.atarimania.com/documents/atari_800XL_Field_Service_manual.pdf) ([http://www.atarimania.com/documents/atari\\_800XL\\_Field\\_Service\\_manual.pdf](http://www.atarimania.com/documents/atari_800XL_Field_Service_manual.pdf)) ]):

Ground	VSS	1		40	D2	Data Bus
Data Bus	D3	2		39	D1	Data Bus
Data Bus	D4	3		38	D0	Data Bus
Data Bus	D5	4		37	AUDIO	Audio Out
Data Bus	D6	5		36	A0	Address Bus
Data Bus	D7	6		35	A1	Address Bus
Phase 2 Clock	Ø2	7		34	A2	Address Bus
Pot Scan	P6	8		33	A3	Address Bus
Pot Scan	P7	9		32	R/W	Read/Write
Pot Scan	P4	10		31	CS1	Chip Select
Pot Scan	P5	11		30	CS0	Chip Select
Pot Scan	P2	12		29	IRQ	Interrupt Request
Pot Scan	P3	13		28	SOD	Serial Output Data
Pot Scan	P0	14		27	ACLK	Serial Output Clock
Pot Scan	P1	15		26	BCLK	Bidirectional Clock
Keyboard Response	KR2	16		25	KRI	Keyboard Response
5 V Power	VCC	17		24	SID	Serial Input Data
Keyboard Scan	K3	18		23	K0	Keyboard Scan
Keyboard Scan	K4	19		22	K1	Keyboard Scan
Keyboard Scan	K5	20		21	K2	Keyboard Scan

From this diagram we can clearly see that the wired pins are K0 to K5, KR0 and KR1 and of course the ground. Following the wires from the PCB to the logic analyzer the channel order is also K0-K5, KR0-1.

So we found the chip and the pinout as well, but how it works? For this the schematics included to the service manual was a huge help for us. The following is the relevant section of the hardware, necessary to understand the reading of the keyboard.



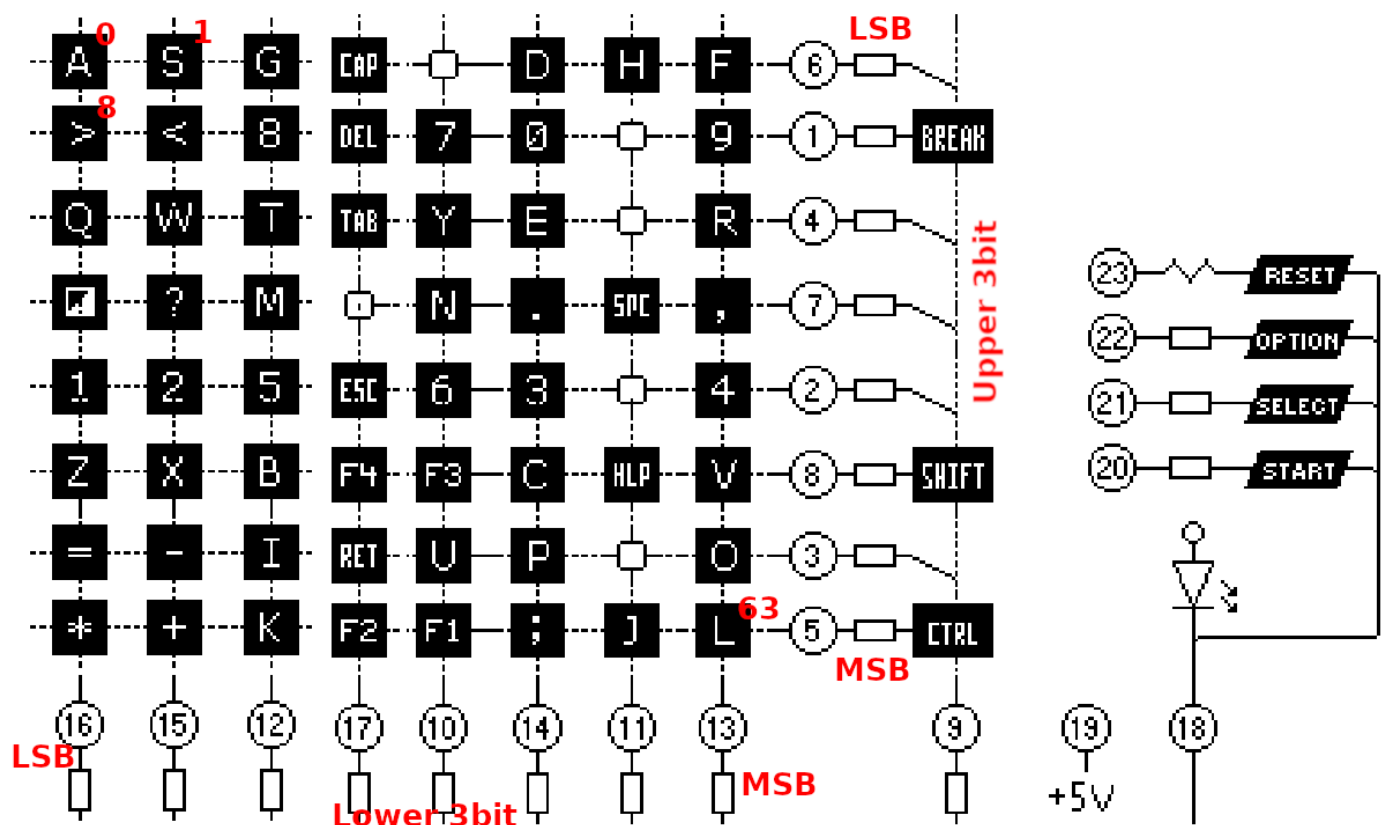


The only good diagram on the keyboard matrix wiring can be found at [ [http://members.casema.nl/hhaydn/howel/logic/burchd/b5\\_800xl\\_kbd.htm](http://members.casema.nl/hhaydn/howel/logic/burchd/b5_800xl_kbd.htm) ([http://members.casema.nl/hhaydn/howel/logic/burchd/b5\\_800xl\\_kbd.htm](http://members.casema.nl/hhaydn/howel/logic/burchd/b5_800xl_kbd.htm)) ] .Still it orders the wires by pin count, where ordering by multiplex selectors would be much more understandable! So here is the reordered image which corresponds to the multiplexer connections (and thus the adjacent rows/columns are really has teh difference of 1).

On the upper diagram with purple the current path is noted in case of pressing the 'E' key.

The modifier keys can be read via the KR2 port, and changes its value by the upper half of the 6bit K0-5 counter.

## ATARI XL Keyboard Matrix (F1/F4 = 1200XL only) Fox-1 1999



So now, understanding the underlying hardware working, let's turn to the measurement CSV file. Probably it was recorded with the Saleae Logic Analyzator and exported to CSV format, but unfortunately the same program cannot import from CSV... We attempted to process it in python.

In [436]:

```
1 %matplotlib inline
2
3 import matplotlib
4 import matplotlib.pyplot as plt
5 import scipy as sp
6 plt.style.use("seaborn-whitegrid")
7 import pandas as pd
8 from ipywidgets import interact, interactive, fixed, interact_manual
9 import ipywidgets as widgets
10 from bisect import bisect_left
```

Import the data and rename those unmanageable column names

In [444]:

```
1 dat = pd.read_csv(  
2     'data.csv',  
3     names=['ta', 'a6', 'a7', 't0', 'd0', 't1', 'd1', 't2', 'd2', 't3', 'd3', 't4', 'd4', 't5',  
4     delimiter="," ,  
5     skipinitialspace=True,  
6     skiprows=1,  
7 )
```

In the imported data the columns don't have equal height, create simple scipy arrays from the continuous data (drop N/A fields)

In [445]:

```
1 t, d = [], []  
2 for i in range(8):  
3     t.append(sp.array(list(dat["t" + str(i)].dropna())))  
4     d.append(sp.array(list(dat["d" + str(i)].dropna())))  
5 t.append(sp.array(list(dat["ta"].dropna())))  
6 t.append(sp.array(list(dat["ta"].dropna())))  
7 d.append(sp.array(list(dat["a6"].dropna())))  
8 d.append(sp.array(list(dat["a7"].dropna())))  
9 del dat # free up some memory
```

For managing the waveform, let's create a helper function which can plot the waveform in time domain. The measurement contained the data in delta-format: only the time of changes are saved for the digital data, analog data has evenly sampled values.

In [446]:

```
1 def plot_analizator(a=0.0, b=0.01, title=""):
2     ax, subplots = plt.subplots(10, 1, figsize=(20, 10))
3     if title:
4         ax.suptitle(title)
5     ax.subplots_adjust(hspace=0)
6
7     for i in range(10):
8         if i < 8:
9             # digital signals
10            subplots[i].step(
11                # +-1 to have the at least 3 points to be able to connect them
12                t[i][bisect_left(t[i], a)-1:bisect_left(t[i], b)+1],
13                d[i][bisect_left(t[i], a)-1:bisect_left(t[i], b)+1],
14                where='post',
15            )
16            subplots[i].axis(xmin=a, xmax=b, ymin=-0.1, ymax=1.1)
17        else:
18            # analog signals
19            subplots[i].plot(
20                # +-1 to have the at least 3 points to be able to connect them
21                t[i][bisect_left(t[i], a)-1:bisect_left(t[i], b)+1],
22                d[i][bisect_left(t[i], a)-1:bisect_left(t[i], b)+1],
23                '-.-',
24            )
25            subplots[i].axis(xmin=a, xmax=b, ymin=-0.1, ymax=5.1)
26
27            subplots[i].set_yticks([])
28            if i != 9:
29                subplots[i].xaxis.set_ticklabels([])
30            else:
31                subplots[i].xaxis.get_major_formatter().set_useOffset(False)
32            if (i < 6):
33                subplots[i].set_ylabel("K%d"%i)
34            else:
35                subplots[i].set_ylabel("KR%d"%(((i-6)&1)+1))
36
```

Now let's validate the information what we could gather so far based on the excellent service manuals:

- K0-5 are indeed act like a counter (even though there is some offset between the changes)
- KR1 and 2 are low active and remains low during a K0 pulse-width time

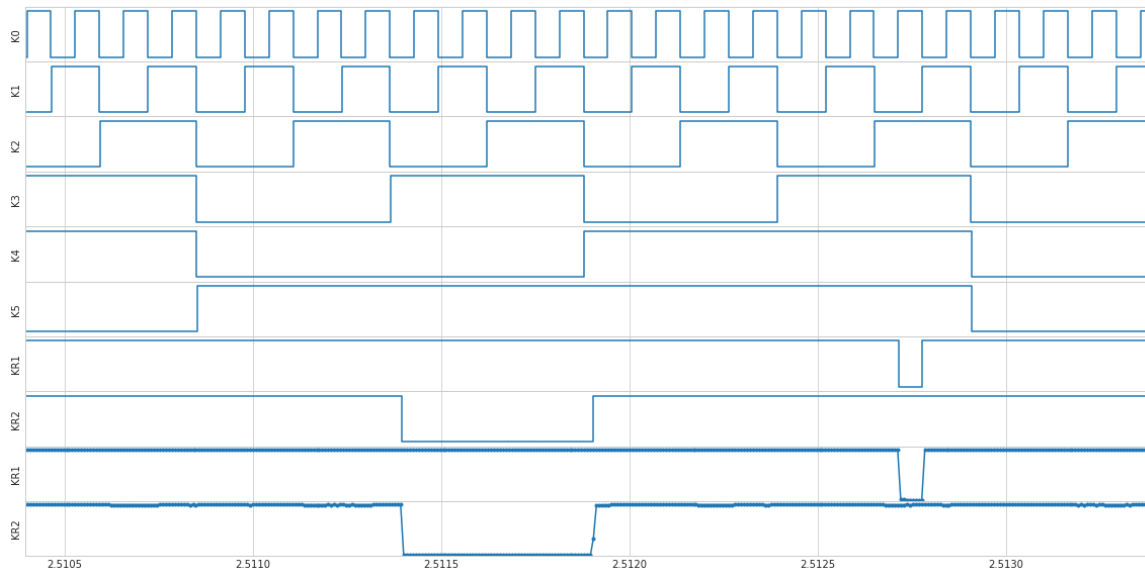
Because the logic analyzer only saved the change of the signal, and the data is binary, thus it is enough to take every two samples offsetted by one to always sample the low signal values on the KR1 and 2 pins.

In [429]:

```
1 @interact(n=(0, len(t[7][1::2])-1, 1))
2 def nth_press_key(n):
3     t_press = t[7][1::2][n]
4     plot_analizator(t_press-0.001, t_press+0.002, title="The %dth key press of
```

n

The 2th key press of a modifier key



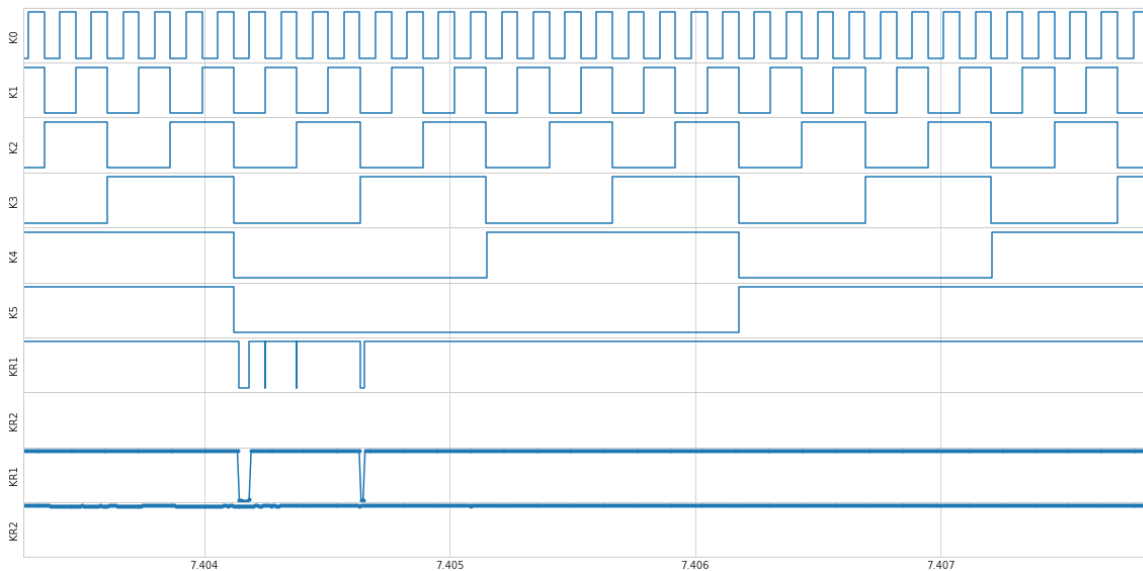
Unfortunately there are some glitches in the digital and in the analog waveform as well. There are some spikes, occurring only for the duration of a sample time (which probably was 10MSps). But there are other, little longer signal, which not sustain its value for the whole K0 pulse duration: that is the bouncing of the keyboards.

In [448]:

```
1 @interact(n=(0, len(t[6][2::1])-1, 1), dt=(0.0001,0.01,0.0005))
2 def nth_press_key(n, dt):
3     t_press = t[6][2::1][n]
4     plot_analizator(t_press-dt/5, t_press+dt*4/5, title="The %dth key press of
```

n  248  
dt  0.00

The 248th key press of a letter

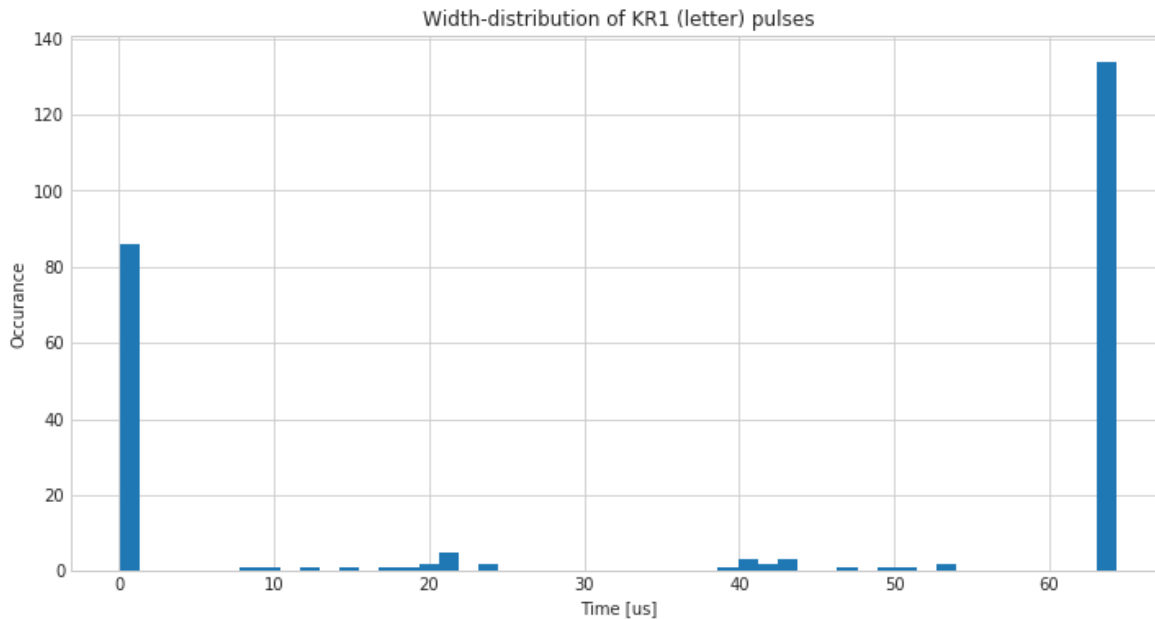


Those glithes must be filtered! The most simple filtering method is the pulse-width based filtering. For this we have to compute the corrent pulse-width and somehow have to separate from the glithes. Using a histogram of the adjacent edges it can be easily seen that most of the pulses are around 65usec, but also there are a few around 40usec and the clearly glithes are below 20usec. Let's use 30usec as a separator: any pulse below 30usec will be removed.



In [449]:

```
1 plt.figure(figsize=(12, 6))
2 plt.title("Width-distribution of KR1 (letter) pulses")
3 plt.ylabel("Occurance")
4 plt.xlabel("Time [us]")
5 plt.hist(sp.diff(t[6])[1::2]*1e6, bins=50);
```



In [450]:

```
1 # so a pulse must be longer than 30usec
2 def get_filtered_KR1(t, t_threashold=30e-6):
3     diffs_ok = sp.diff(t)[1::2] > t_threashold
4     return sp.array([t[2*i+1] for i, isok in enumerate(diffs_ok) if isok])
5 t_filt = get_filtered_KR1(t[6])
```

Now we can move on to the actual key decode phase. Firstly a table must be created for the keyscan codes. This array is constructed based on the previous keyboard matrix wiring. The shifted key (when SHIFT is pressed) is also included.

In [413]:

```
1 keyscans = [  
2     'a', 's', 'g', '<CAP>', '<NC>', 'd', 'h', 'f',  
3     '>', '<', '8', '<DEL>', '7', '0', '<NC>', '9',  
4     'q', 'w', 't', '<TAB>', 'y', 'e', '<NC>', 'r',  
5     '<???>', '?', 'm', '<NC>', 'n', '.', ' ',  
6     '1', '2', '5', '<ESC>', '6', '3', '<NC>', '4',  
7     'z', 'x', 'b', '<F4>', '<F3>', 'c', '<HLP>', 'v',  
8     '=', '-', 'i', '<RET>', 'u', 'p', '<NC>', 'o',  
9     '*', '+', 'k', '<F2>', '<F1>', ';', 'j', 'l'  
10 ]  
11  
12 keyscans_shift = [  
13     'A', 'S', 'G', '<CAP>', '<NC>', 'D', 'H', 'F',  
14     '>', '<', '@', '<DEL>', '?', ')', '<NC>', '(',  
15     'Q', 'W', 'T', '<TAB>', 'Y', 'E', '<NC>', 'R',  
16     '<???>', '?', 'M', '<NC>', 'N', ']', ' ', '[',  
17     '!', '"', '%', '<ESC>', '&', '#', '<NC>', '$',  
18     'Z', 'X', 'B', '<F4>', '<F3>', 'C', '<HLP>', 'V',  
19     '|', '_', 'I', '<RET>', 'U', 'P', '<NC>', 'O',  
20     '^', '\\', 'K', '<F2>', '<F1>', ':', 'J', 'L'  
21 ]
```

And finally the phase of decoding the keys!

As there is a little offset among K0-5 and between KR1, let's not rely on the K0-5 numbers (or mindig with the correction of those values)! Instead let's replace K0-5 entirely by a 6bit counter. This counter is "clocked" by the edge changes on K0 and has an initial value. So the K0-5 value is known at any time, just count up until that time the K0 changes (which is the index of the current time) and add the initial value, finally take a modulo 64 opetarion. This time the time->index mapping is solved by a bisection search on the array of time values.

So going over the filtered 1->0 change of KR1 we can get the sensed key(s). The modifier keys can also be sensed, comparing the timestamp of the closest modkey press-down with the current letter press-down: if they are very close, the modifier was pressed for the letter key.

A very dump key repetition blocking algorithm was also implemented to avoid the repetatedly sensed keys (and fortunately flag doesn't have repeated letter in it).

In [452]:

```
1 # first value is 0, and bisect search will gave 1 for t=0, so subtract 1 in ad
2 initial_val = int(sum([d[i][0]*(1<i) for i in range(6)]))-1
3 old_char = '\x00'
4
5 for t_keypress in t_filt:
6     scancode = (initial_val+bisect_left(t[0], t_keypress)) & 0x3f
7     modkey_near_time = t[7][1::2][min(bisect_left(t[7][1::2], t_keypress), len(
8
9     if sp.absolute(modkey_near_time - t_keypress) < 1e-1:
10         modkey = ((initial_val+bisect_left(t[0], modkey_near_time)) & 0x3f) >>
11         if modkey == 5: # Shift
12             curr_char = keyscans_shift[scancode]
13         else:
14             curr_char = '<MOD???>'
15     else:
16         curr_char = keyscans[scancode]
17
18     if curr_char != old_char:
19         print(old_char, end='')
20     old_char = curr_char
```

flag: 8-bit-hardware-keylogger

So finally we here is the flag!